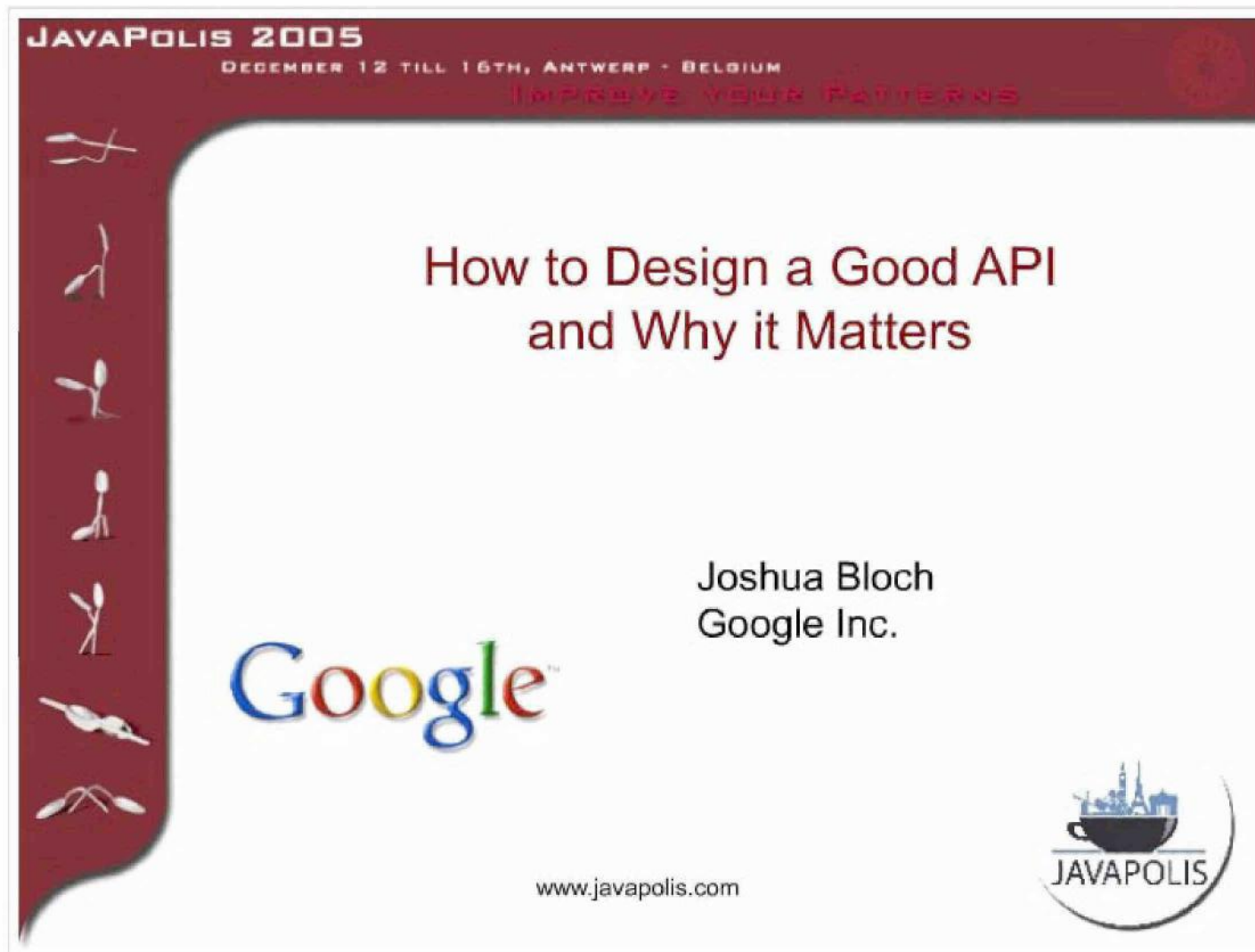


Bush Declaration Ex. Q

TX 0624



UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA

TRIAL EXHIBIT 624

CASE NO. 10-03561 WHA

DATE ENTERED _____

BY _____

DEPUTY CLERK

JAVAPOLIS 2005

IMPROVE YOUR PATTERNS

Why is API Design Important?

- APIs can be among a company's greatest assets
 - Customers invest heavily: buying, writing, learning
 - Cost to stop using an API can be prohibitive
 - Successful public APIs capture customers
- Can also be among company's greatest liabilities
 - Bad APIs result in unending stream of support calls
- Public APIs are forever - one chance to get it right



www.javapolis.com

OAGOOGLE0100219512

JAVAPOLIS 2005

IMPROVE YOUR PATTERNS

Why is API Design Important to *You*?

- If you program, you are an API designer
 - Good code is modular—each module has an API
- Useful modules tend to get reused
 - Once module has users, can't change API at will
 - Good reusable modules are corporate assets
- Thinking in terms of APIs improves code quality



www.javapolis.com

OAG00GLE0100219513

JAVAPOLIS 2005

IMPROVE YOUR PATTERNS

Characteristics of a Good API

- Easy to learn
- Easy to use, even without documentation
- Hard to misuse
- Easy to read and maintain code that uses it
- Sufficiently powerful to satisfy requirements
- Easy to extend
- Appropriate to audience



www.javapolis.com

OAG00GLE0100219514

JAVAPOLIS 2005

IMPROVE YOUR PATTERNS

Outline

- I. The Process of API Design
- II. General Principles
- III. Class Design
- IV. Method Design
- V. Exception Design
- VI. Refactoring API Designs




www.javapolis.com

OAG00GLE0100219515

JAVAPOLIS 2005
IMPROVE YOUR PATTERNS

I. The Process of API Design

www.javapolis.com

OAGOOGLE0100219516

JAVAPOLIS 2005

IMPROVE YOUR PATTERNS

Gather Requirements—with a Healthy Degree of Skepticism

- Often you'll get proposed solutions instead
 - Better solutions may exist
- Your job is to extract true requirements
 - Should take the form of **use-cases**
- Can be easier and more rewarding to build something more general

 Good



www.javapolis.com

OAG00GLE0100219517

JAVAPOLIS 2005

IMPROVE YOUR PATTERNS

Start with Short Spec—1 Page is Ideal

- At this stage, agility trumps completeness
- Bounce spec off as many people as possible
 - Listen to their input and take it seriously
- If you keep the spec short, it's easy to modify
- Flesh it out as you gain confidence
 - This necessarily involves coding



www.javapolis.com

OAGOOGLE0100219518

JAVAPOLIS 2005

IMPROVE YOUR PATTERNS

Write to Your API Early and Often

- Start *before* you've implemented the API
 - Saves you doing implementation you'll throw away
- Start *before* you've even specified it properly
 - Saves you from writing specs you'll throw away
- Continue writing to API as you flesh it out
 - Prevents nasty surprises
 - **Code lives on as examples, unit tests**



www.javapolis.com

OAGOOGLE0100219519

JAVAPOLIS 2005

IMPROVE YOUR PATTERNS

Writing to SPI is Even More Important

- Service Provider Interface (SPI)
 - Plug-in interface enabling multiple implementations
 - Example: Java Cryptography Extension (JCE)
- Write multiple plug-ins before release
 - If one, it probably won't support another
 - If two, it will support more with difficulty
 - If three, it will work fine
- Will Tracz calls this "The Rule of Threes"
(*Confessions of a Used Program Salesman*, Addison-Wesley, 1995)



Bad

www.javapolis.com

OAG00GLE0100219520

JAVAPOLIS 2005

IMPROVE YOUR PATTERNS

Maintain Realistic Expectations

- Most API designs are over-constrained
 - You won't be able to please everyone
 - Aim to displease everyone equally
- Expect to make mistakes
 - A few years of real-world use will flush them out
 - Expect to evolve API




www.javapolis.com

OAG00GLE0100219521

JAVAPOLIS 2005
IMPROVE YOUR PATTERNS

II. General Principles

www.javapolis.com

OAGOOGLE0100219522

JAVAPOLIS 2005

IMPROVE YOUR PATTERNS

API Should Do One Thing and Do it Well

- Functionality should be easy to explain
 - If it's hard to name, that's generally a bad sign
 - Good names drive development
 - Be amenable to splitting and merging modules



www.javapolis.com

OAGOOGLE0100219523

JAVAPOLIS 2005

IMPROVE YOUR PATTERNS

API Should Be As Small As Possible But No Smaller

- API should satisfy its requirements
- **When in doubt leave it out**
 - Functionality, classes, methods, parameters, etc.
 - **You can always add, but you can never remove**
- *Conceptual weight* more important than bulk
- Look for a good *power-to-weight ratio*



www.javapolis.com

OAG00GLE0100219524

JAVAPOLIS 2005

IMPROVE YOUR PATTERNS

Implementation Should Not Impact API

- Implementation details
 - Confuse users
 - Inhibit freedom to change implementation
- Be aware of what is an implementation detail
 - Do not overspecify the behavior of methods
 - For example: **do not specify hash functions**
 - All tuning parameters are suspect
- Don't let implementation details “leak” into API
 - On-disk and on-the-wire formats, exceptions



www.javapolis.com

OAGOOGLE0100219525

JAVAPOLIS 2005

IMPROVE YOUR PATTERNS

Minimize Accessibility of Everything

- Make classes and members as private as possible
- Public classes should have no public fields (with the exception of constants)
- This maximizes **information hiding**
- Allows modules to be used, understood, built, tested, and debugged independently



www.javapolis.com

OAGOOGLE0100219526

JAVAPOLIS 2005

IMPROVE YOUR PATTERN

Names Matter—API is a Little Language

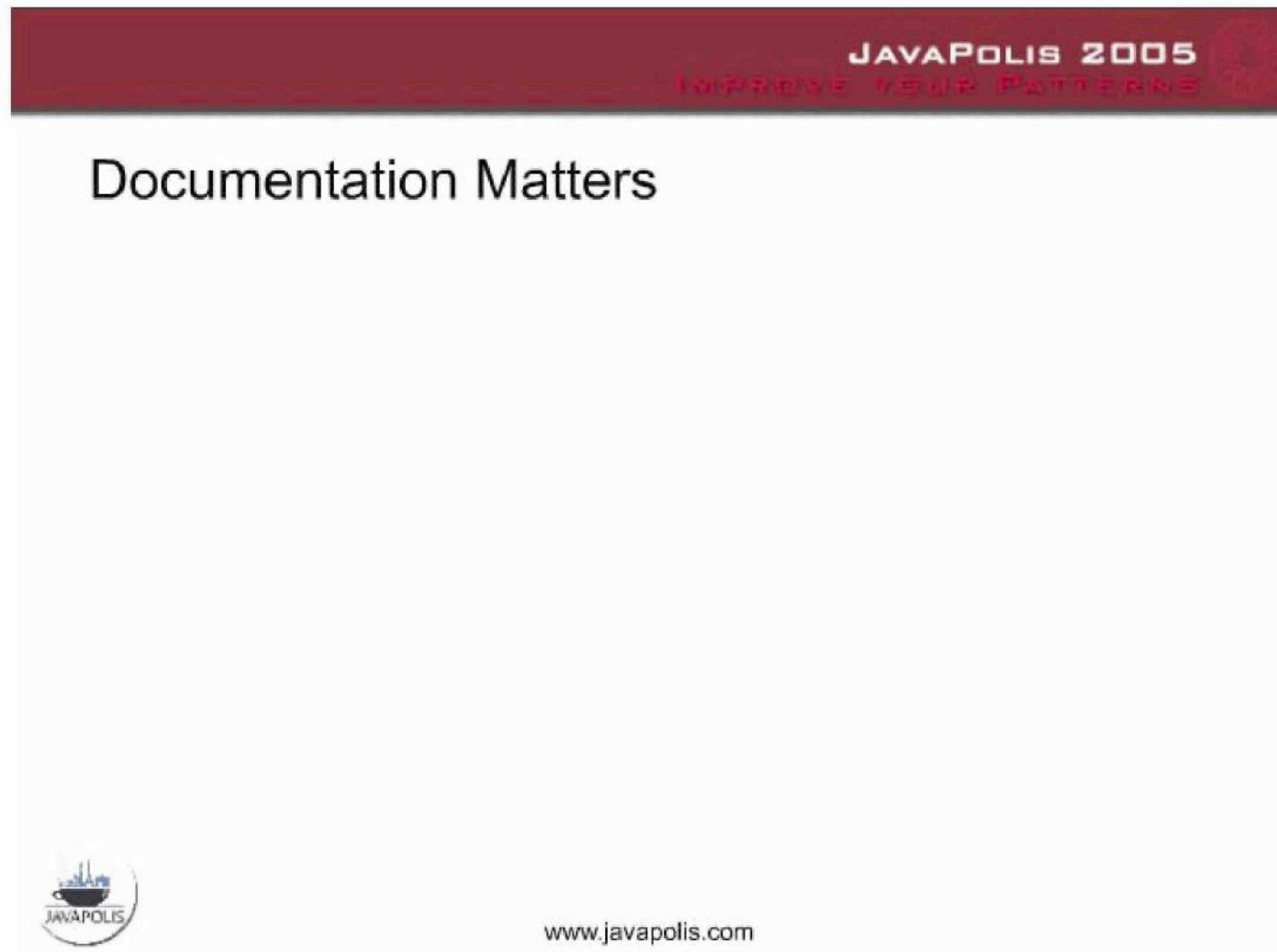
- Names Should Be Largely Self-Explanatory
 - Avoid cryptic abbreviations
- Be consistent—same word means same thing
 - Throughout API, (Across APIs on the platform)
- Be regular—strive for symmetry
- Code should read like prose

```
if (car.speed() > 2 * SPEED_LIMIT)
    generateAlert("Watch out for cops!");
```



www.javapolis.com

OAG00GLE0100219527



OAGOOGLE0100219528

JAVAPOLIS 2005

IMPROVE YOUR PATTERNS

Document Religiously

- Document **every** class, interface, method, constructor, parameter, and exception
 - Class: what an instance represents
 - Method: contract between method and its client
 - Preconditions, postconditions, side-effects
 - Parameter: indicate units, form, ownership
- Document state space very carefully



www.javapolis.com

OAGOOGLE0100219529

JAVAPOLIS 2005

IMPROVE YOUR PATTERNS

Consider Performance Consequences of API Design Decisions

- Bad decisions can limit performance
 - Making type mutable
 - Providing constructor instead of static factory
 - Using implementation type instead of interface
- Do not warp API to gain performance
 - Underlying performance issue will get fixed, but headaches will be with you forever
 - Good design usually coincides with good performance



www.javapolis.com

OAG00GLE0100219530

JAVAPOLIS 2005

IMPROVE YOUR PATTERNS

Effects of API Design Decisions on Performance are Real and Permanent

- `Component.getSize()` returns `Dimension`
- `Dimension` is mutable
- Each `getSize` call must allocate `Dimension`
- Causes *millions* of needless object allocations
- Alternative added in 1.2; old client code still slow



www.javapolis.com

OAGOOGLE0100219531

JAVAPOLIS 2005

IMPROVE YOUR PATTERNS

API Must Coexist Peacefully with Platform

- Do what is customary
 - Obey standard naming conventions
 - Avoid obsolete parameter and return types
 - Mimic patterns in core APIs and language
- Take advantage of API-friendly features
 - Generics, varargs, enums, default arguments
- Know and avoid API traps and pitfalls
 - Finalizers, public static final arrays
- **Don't Transliterate APIs**



www.javapolis.com

OAG00GLE0100219532

JAVAPOLIS 2005
Interactive Java Patterns

III. Class Design

www.javapolis.com

OAGOOGLE0100219533